INTE 21233
Web Applications Development - 1

Final Project – Documentation

Group - 8

- *IM/2021/116 – KITCHILAN F.N.*
- *IM/2021/081 – RATNAYAKA R.M.H.N.*
- *IM/2021/063 – WADITHYA G.S.*
- *IM/2021/051 – DE SILVA D.G.M*
- *IM/2021/045 – MATHUMITHAN M.*
- *IM/2021/039 – FONSEKA S.A.*
- *IM/2021/031 – JAYASURIYA D.D.*
- *IM/2021/007 – SAMARAKKODI S.D.S.H.*

Table of Contents

header.php

1. Purpose of the Code

The code is for the setup of a basic web application called "SwiftStock". This application is designed to help users manage their inventory. The code sets up the structure and appearance of the web page using various technologies and libraries. It ensures that the page is styled correctly, responsive on different devices, and equipped with necessary functionalities to support the inventory management system.

2. Functionalities

Here's what each part of the code is responsible for:

- **Page Title and Icon**:
    - The page title is set to "SwiftStock", which appears on the browser tab.
    - An icon (logo) is displayed in the browser tab, which helps users identify the web page quickly.
- **Styling**:
    - **Bootstrap CSS**: Adds pre-designed styles and components to make the page look good and work well on different devices.
    - **Custom CSS Files**: Additional styles for the navigation bar, footer, and landing page to give the web page a unique look and feel.
    - **Internal CSS**: Styles specific to the page, such as font sizes for headings and spacing, to ensure that content is displayed neatly.
- **Responsive Design**:
    - Ensures that the web page adjusts its layout and font sizes based on the screen size of the device being used (like a phone, tablet, or computer).
- **JavaScript Libraries and Files**:
    - **Bootstrap JS**: Adds interactive features like modals, dropdowns, and tooltips.
    - **jQuery**: Provides additional functionality to handle events, animations, and DOM manipulation.
    - **Popper.js**: Used by Bootstrap to position pop-ups and tooltips correctly.
    - **Chart.js**: Helps in creating charts if the application needs to display graphical data.
    - **Custom JavaScript Files**: Scripts for additional functionalities specific to the inventory management system, like form validations or dynamic content updates.

3. How Does the Code Work?

Here's a breakdown of how the code components work together:

- **Head Section**:
    - **Meta Tags**: Define the character encoding (UTF-8) and make the page responsive by adjusting to different screen sizes.
    - **Title and Icon**: Sets the title and favicon of the page, so users can easily recognize the tab.
    - **Stylesheets**:
        - **Bootstrap CSS**: Applies a basic layout and style, making it easier to design the page without writing too much custom CSS.
        - **Custom CSS**: Adds specific styles for different parts of the page, like the navigation bar and footer.
        - **Internal CSS**: Directly inside the HTML, this provides quick and easy styling adjustments for headings and other elements.
    - **JavaScript Files**:
        - **Bootstrap JS and Popper.js**: These scripts add interactive components (like drop-down menus) and manage their positioning.
        - **jQuery**: Simplifies JavaScript tasks, making it easier to handle things like user interactions.
        - **Chart.js**: If the application needs to show charts, this library will help render them.
        - **Custom JavaScript**: Contains your own code for specific functionalities related to the inventory management, such as validating user input or handling dynamic updates on the page.

footer.php

1. Purpose of the Code

The code is for the **footer** section of our web application. The footer is the area that appears at the bottom of the web page. It usually contains important information and links that users may find helpful or necessary. In this case, the footer provides information about our site, legal notices, and a way for users to subscribe to a newsletter.

2. Functionalities

Here’s what each part of the footer does:

- **Info Section**:
    - Contains links to important pages such as "About Us", "Customers", and "Service". This helps users navigate to different sections of the website or related external sites.
- **Explore Section**:
    - Provides links to pages like "Why Us", "Latest Features", and "New Uploads". This section is intended to guide users to more information about the company, its features, and updates.
- **Legal Section**:
    - Includes links to legal documents such as the "Service Level Agreement", "Privacy Policy", and "Terms and Conditions". These links are crucial for users to understand their rights and the rules governing their use of the website.
- **Newsletter Subscription**:
    - Allows users to subscribe to a newsletter by entering their email address. This helps keep users informed about news, updates, and special offers.
- **Social Media Icons**:
    - Provides icons for social media platforms (Facebook, Twitter, LinkedIn, GitHub). These icons are clickable and link to your social media profiles, helping users connect with you on different platforms.
- **Copyright Notice**:
    - Displays a copyright statement with the current year and the name of your company. This is important for legal reasons and to show that you hold the rights to the content on your site.

3. How Does the Code Work?

Here's a step-by-step explanation of how each part of the code works:

- **HTML Structure**:
    - The footer is divided into several columns ( ). Each column contains different types of information and links.
- **Info Section**:
    - Uses a list (<ul>) to organize links under a heading ("Info"). Each link is listed as an item (<li>) and directs users to various pages or external sites.
- **Explore Section**:
    - Like the Info section, this section also uses a list to provide links to different areas of interest or recent updates.
- **Legal Section**:

- Contains links to important legal documents. Each link opens in a new tab (target="blank"), ensuring that users don't leave your site when accessing these documents.

- **Newsletter Subscription**:
  - Includes a form with an input field for users to enter their email and a submit button. When users fill out the form and click the button, their email is submitted for subscription (though the actual submission functionality would need a backend script to process this).
- **Social Media Icons**:
  - Uses icons from FontAwesome (a popular icon library). Each icon represents a social media platform and is clickable to take users to your social media pages.
- **Copyright Notice**:
  - Dynamically displays the current year using PHP, which is a server-side scripting language. This ensures the copyright year is always up-to-date without manual changes.
- **Bootstrap JavaScript**:
  - The script tag at the end includes Bootstrap's JavaScript library. Bootstrap's JavaScript adds interactive features and functionality to your website. This file is essential for making sure that any interactive Bootstrap components (like dropdowns or modals) work properly.

navigation.php

1. Purpose of the Code

The purpose of this code is to create a **navigation bar** for our web application. This navigation bar will appear at the top of your web pages and allow users to easily navigate between different sections of our inventory management system. It includes links to various pages, displays the user's name, and adapts to different screen sizes using a collapsible menu for smaller devices.

2. Functionalities

Here's a breakdown of the functionalities provided by the navigation bar code:

- **User Greeting**: Displays the user's name in the navigation bar. This is pulled from the user's session data and sanitized for security.
- **Navigation Links**: Provides links to important pages such as "Home", "Manage Inventory", "Search Inventory", "Create Reports", "About Us", and "My Account". These links allow users to navigate to different parts of the web application.

- **Active Page Highlighting**: Highlights the link corresponding to the current page, so users know which page they are currently on.
- **Responsive Design**: Uses Bootstrap's offcanvas menu to create a navigation menu that adapts to different screen sizes. On smaller screens, the menu collapses into a side panel that can be toggled open or closed.

3. How Does the Code Work?

Here's how the code works in detail:

- **PHP Section**:
    - **Session Handling**: The PHP code checks if the user's full name is stored in the session. It then sanitizes this name to prevent security issues like cross-site scripting (XSS). If the name is not set, it defaults to 'Name'.
    - **Current Page Detection**: It determines the name of the current page (like home.php or manage_inventory.php) so that it can set the correct navigation link as "active".
    - **Include Header**: It includes a separate header component (Components/header.php) which likely contains other common elements for the web pages.
- **HTML and Bootstrap**:
    - **Navbar Structure**:
        - **Branding**: Displays the site's logo and name. This section uses an image for the logo and text for the site name.
        - **Offcanvas Menu**:
            - **Header**: Shows the user's avatar and name at the top of the offcanvas menu. There's also a button to close the menu.
            - **Navigation Links**: Lists the main navigation links. The PHP code adds an "active" class to the link of the current page, making it visually distinct from the others.
        - **Toggler Button**: This button appears on smaller screens and allows users to open or close the offcanvas menu. It ensures that the navigation bar remains usable even on mobile devices.

index.php

1. Purpose of the Code

The purpose of the code is to build the **landing page** of a web application named "SwiftStock." This landing page serves as the initial interface that users see when they visit the application. It introduces

the application, explains its features, and provides navigation to other parts of the site. Additionally, it displays customer reviews and various sections that describe the services and benefits of using the application.

2. Functionalities

Here are the key functionalities of the code:

- **Navbar Creation**: A navigation bar is created at the top of the page for easy access to other pages and functionalities.
- **Hero Section**: An engaging hero section is included that welcomes users and provides a call-to-action button.
- **Why Us Section**: This section explains why users should choose "SwiftStock" by highlighting its benefits and features.
- **Services Section**: The services provided by "SwiftStock" are showcased with cards explaining different functionalities.
- **Reviews Section**: Customer reviews are displayed in a carousel format to provide social proof and feedback on the application.
- **Footer Inclusion**: A footer is included at the bottom of the page to provide additional navigation and information.

3. How Does the Code Work?

Here is a step-by-step explanation of how the code functions:

1. **PHP Section**:
    - The require_once 'Components/header.php' statement includes a PHP file that likely contains common HTML code for the header of the page. This file is included to ensure that the header appears on this page as well as any other pages that include it.
2. **Navigation Bar**:
    - A <nav> element is used to define the navigation bar. It is given the classes navbar, navbar-expand-lg, and bg-body-tertiary to apply Bootstrap styling and make it responsive.
    - Inside the <nav> element, a brand logo and name are displayed. The <button id="loginbtn"> Log In </button> is also included for users to log into their accounts.
3. **Hero Section**:
    - The <section id="hero" class="hero section dark-background"> defines the hero section with a dark background.
    - This section includes a welcome message with a typewriter effect created using data attributes (data-period, data-type) and a call-to-action button labeled "Get Started!"

- An image is displayed alongside the text to enhance visual appeal.

4. **Why Us Section**:
    - The section is introduced with the heading "Why SwiftStock" and contains three cards. Each card describes different aspects of the application:
        - **Card 1**: Describes SwiftStock as a cloud-based inventory management system.
        - **Card 2**: Lists benefits such as real-time updates and automation.
        - **Card 3**: Highlights features like tracking and report generation.
5. **Services Section**:
    - This section presents the services offered by SwiftStock with individual cards. Each card includes an image and a description of the service:
        - **Card 1**: Describes inventory management.
        - **Card 2**: Describes search functionality.
        - **Card 3**: Describes report creation.
6. **Reviews Section**:
    - The <section id="reviews"> introduces a section for customer reviews.
    - An Owl Carousel is used to create a sliding effect for displaying testimonials. Each testimonial includes a customer's photo, name, and their review of the product.
7. **Footer Inclusion**:
    - The <?php require_once 'Components/footer.php' ?> statement includes a footer file at the bottom of the page. This footer is used for additional navigation and information.

main.js

1. Purpose of the Code

The purpose of the code is to add interactive and dynamic elements to the web application. Specifically, it:

- Creates a typewriter effect for text elements on the page.
- Adds animations to elements as they scroll into view.
- Handles button clicks for navigation and form interactions.

2. Functionalities

Here are the key functionalities provided by the code:

1. **Typewriter Animation**:
    - Text on the page is animated to appear as if it is being typed out in real-time.
2. **On-Scroll Animations**:
    - Elements that are initially hidden will fade into view as the user scrolls down the page.
3. **Button Click Handlers**:
    - Buttons are set up to perform specific actions when clicked, such as redirecting to different pages or clearing form fields.

3. How Does the Code Work?

1. Typewriter Animation:

- A TxtType constructor function is used to create the typewriter effect.
    - **Initialization**: The function takes parameters for the element to apply the effect to, the array of strings to rotate through, and the duration of each typing cycle.
    - **Tick Function**: This function handles the typing and deleting effects by updating the displayed text. It adds or removes characters to simulate typing and deleting.
    - **Timing**: The typing speed varies randomly, and the text pauses briefly when it finishes typing before starting to delete. The function continuously calls itself to update the text at intervals.
    - **Initialization on Page Load**: When the page loads, the script looks for elements with the class typewrite, gets the text and duration attributes from these elements, and applies the typewriter effect to them. Additionally, a CSS style is added to create a cursor effect for the typewriter animation.

2. On-Scroll Animations:

- An IntersectionObserver is used to detect when elements with the class hidden-cls come into view.
    - **Observation**: The observer watches these elements and adds a show class to them when they are scrolled into view. This show class is typically used to apply animations or make the elements visible.

3. Button Click Handlers:

- **Navigation Buttons**: Event listeners are added to buttons to redirect users to different pages when clicked:
    - **Get Started Button**: Redirects users to the register.php page.
    - **Login Button**: Redirects users to the login.php page.
    - **Login Button in a Different Context**: Redirects users to the home.php page.

- **Clear Form Button**: Clears the values of form fields with IDs id, name, and category when clicked. This is useful for resetting forms.

verification.js

1. Purpose of the Code

The purpose of this code is to ensure that the data entered into various forms on the web application is correct and complete before it is submitted. This helps in preventing errors and ensures that only valid data is processed.

2. Functionalities

The code provides validation for several forms within the application:

1. **Registration Form**:
    - **Name**: Checks if the name is provided and has a valid format.
    - **Username**: Ensures that the username is at least 3 characters long and follows the correct format.
    - **Email**: Validates the email address to ensure it is in a correct format.
    - **Telephone Number**: Verifies that the telephone number is provided, starts with '0', and does not contain too many repeated digits.
    - **Password and Confirm Password**: Ensures that the password meets security requirements and that the confirmation password matches the original.
2. **Item and Category Forms**:
    - **Add Item**: Validates item name, quantity, price, and description.
    - **Add Category**: Checks that the category name is valid.
    - **Update Item**: Validates item ID and the new details such as name, quantity, price, and description.
    - **Delete Item**: Ensures that either an ID or name is provided to identify the item to be deleted.

3. How Does the Code Work?

1. Registration Form Validation:

- **Event Listener**: An event listener is added to the registration form. This listener triggers when the form is submitted.
- **Validation Checks**:

- **Name**: The code checks if the name is not empty and matches a pattern that allows only letters and spaces.
- **Username**: It ensures the username is at least 3 characters long and only contains letters and numbers.
- **Email**: The email must match a pattern that represents a valid email format.
- **Telephone Number**: The telephone number must be non-empty, start with '0', not contain repeated digits, and consist only of numbers.
- **Password**: The password must be at least 8 characters long, containing at least one uppercase letter, one lowercase letter, and one number.
- **Confirm Password**: The confirmation password must match the original password.

- **Form Submission**: If any validation fails, an error message is displayed, and the form submission is prevented. If all validations pass, the form is submitted.

2. Item and Category Form Validation:

- **Event Listener**: Event listeners are added to the forms related to adding, updating, and deleting items, as well as adding categories. These listeners are triggered when the respective forms are submitted.
- **Add Item Form**:
  - **Name**: The item name must contain only letters and spaces.
  - **Quantity**: Must be a positive number.
  - **Price**: Must be a positive number with up to two decimal places.
  - **Description**: This field is required.
- **Add Category Form**:
  - **Category Name**: Must contain only letters and spaces.
- **Update Item Form**:
  - **ID**: The item ID must be a positive number.
  - **New Details**: Validates the new item details, such as name, quantity, price, and description, if they are provided.
- **Delete Item Form**:
  - **ID or Name**: Either the item ID or name must be provided to identify which item to delete.

General Workflow:

- **Validation**: For each form, the code performs checks to ensure the data entered is valid. If the data does not meet the required conditions, appropriate error messages are shown, and form submission is prevented.
- **Error Messages**: Error messages are displayed to guide users in correcting their input.
- **Form Submission**: The form is only submitted if all validation checks are successful. Otherwise, the submission is blocked, allowing users to correct their input.

This approach ensures that data integrity is maintained and provides users with immediate feedback on their input, improving the overall user experience.

aboutUs.php

1. Purpose of the Code

The code provided is designed to create an "About Us" page for an inventory management web application. This page is intended to inform visitors about the company, its team, and its mission. The code ensures that only logged-in users can access this page and includes both PHP and HTML/CSS to achieve this.

2. Functionalities

- **User Authentication**: The code checks if a user is logged in before displaying the page content. If the user is not logged in, they are redirected to the login page.
- **Page Content**:
    - The page introduces the company, SwiftStock, describing its mission and vision.
    - The team members are showcased with their profiles, including their names, roles, and brief descriptions.
- **Styling and Layout**:
    - The page is styled using CSS to present the content attractively.
    - Flexbox is used for layout purposes, ensuring that elements align properly.
    - Cards are used to display team member information in a neat and organized way.

3. How the Code Works

1. **User Authentication**:
    - The session.php script is included to manage user sessions. This script checks whether the user is logged in.

- If the user is not logged in (!isset($_SESSION['loggedIn'])), they are redirected to the login page using the header() function. This ensures that only authorized users can view the page.

2. **Including Components**:
   - The navigation.php file is included to add a navigation bar to the page. This component provides links to other parts of the website, helping users navigate easily.
3. **Styling with CSS**:
   - CSS is used to style various elements on the page.
     - **Flexbox Layout**: .row class is used to create a flexible layout for rows of content.
     - **Team Member Cards**: The #team-member-card ID styles the cards that contain team member details. It ensures that each card has a consistent look with a white background, rounded corners, and a shadow effect.
     - **Images and Text**: Specific styles are applied to images and text to ensure they are visually appealing. For example, images are rounded and centered, and text is styled with appropriate margins and font sizes.
4. **HTML Structure**:
   - The HTML defines the structure of the page, including:
     - **Introduction Section**: Provides an overview of the company.
     - **Our Story Section**: Shares the background and evolution of the company.
     - **Vision & Mission Section**: Outlines the company's future goals and objectives.
     - **Team Section**: Displays profiles of team members in a grid format. Each profile includes a photo, name, position, and a short description of their role.
5. **Footer**:
   - The footer.php file is included at the end of the page to provide a consistent footer across the website. This may contain contact information, links, or other relevant details.

adddata.php

1. Purpose of the Code

The code is designed to manage inventory items and categories within a web application. It allows users to add new categories and items to the database, while also ensuring that duplicate entries are not created. The code handles user input, interacts with the database, and provides feedback on the success or failure of operations.

2. Functionalities

- **Input Validation**: User inputs are validated and sanitized to prevent incorrect or unsafe data from being processed.
- **Category Management**: Users can add new categories to the database. The system checks if the category already exists before adding it.
- **Item Management**: Users can add new items to the inventory. The system ensures that no duplicate items are added based on their name.
- **Feedback**: After an operation, feedback is provided to the user to indicate whether the action was successful or if there were issues.

3. How the Code Works

1. **Database Connection**:
    - A database connection is established using a separate DBconnection.php file. This file is included at the beginning of the script to ensure that the database connection is available for executing queries.
2. **Input Validation**:
    - A function named validate_input is used to clean and validate user inputs. It removes unnecessary spaces, slashes, and converts special characters to HTML entities to prevent security issues.
    - The function also checks the type of data (string, integer, or float) and ensures it conforms to expected formats.
3. **Category Management**:
    - When a category is being added, the script first validates and sanitizes the input category name.
    - The category_exists function checks if the category already exists in the database. It does this by preparing a query to search for the category name (converted to uppercase to ensure case-insensitivity) and executing it.
    - If the category does not exist, it is inserted into the database. If it already exists, a feedback message is generated to inform the user.
4. **Item Management**:
    - When a new item is added, the script validates and sanitizes all input fields: item name, description, quantity, price, and category ID.
    - The item exists function checks if the item is already present in the database by either its name or ID.

- If the item does not exist, it is added to the database. If an error occurs during the insertion process, or if the item already exists, appropriate feedback is provided to the user.

5. **Feedback and Redirection**:
   - After attempting to add a category or item, the script prepares a feedback message based on the result of the operation.
   - The user is then redirected to the manage_inventory.php page, with the feedback message included in the URL. This message is used to inform the user of the success or failure of their action.
6. **Closing the Database Connection**:
   - After all operations are completed, the database connection is closed to free up resources.

checkLogin.php

1. Purpose of the Code

The purpose of the code is to handle user login for a web application. When a user tries to log in, this code checks whether the entered username and password are correct. If they are, the user is granted access and redirected to the main page of the application. If not, the user is redirected back to the login page with an error message.

2. Functionalities

1. **Input Validation**: The code checks whether the username and password fields are empty and redirects the user if they are.
2. **Database Query**: It queries the database to check if the provided username exists.
3. **Password Verification**: The hashed password stored in the database is compared with the entered password to ensure they match.
4. **Session Management**: If the credentials are correct, a session is started to keep the user logged in and to store their information.
5. **Error Handling and Redirection**: It redirects the user to appropriate pages based on whether the login attempt was successful or if there were errors.

3. How Does the Code Work?

1. **Input Handling**: First, the code checks if the form has been submitted using the POST method. If it has, the username and password from the form are retrieved. These inputs are then sanitized to remove any unnecessary characters or potential security risks.

2. **Empty Field Check**: The code checks if either the username or password is empty. If either field is empty, the user is redirected back to the login page with a message indicating that the fields cannot be empty.
3. **Database Interaction**: If both fields are filled, a SQL query is prepared to search for the username in the database. A prepared statement is used to prevent SQL injection attacks. The query is executed, and the results are checked to see if any user with the given username exists.
4. **Password Verification**: If the username exists, the password entered by the user is compared to the stored hashed password using a function designed for this purpose. If the password matches, the session is started, and user information is saved in the session.
5. **Session Management**: With valid credentials, a session variable is set to indicate that the user is logged in. Another session variable is used to store the user's full name. The user is then redirected to the home page of the application.
6. **Error Handling**: If the password does not match or if no user with the given username is found, the user is redirected back to the login page with an error message. If there is an issue with preparing the SQL statement, an error message is displayed, and the connection is closed.
7. **Closing Resources**: After completing the database operations, the statement and connection are properly closed to free up resources

checkSignup.php

1. Purpose of the Code

The purpose of the code is to handle user registration for a web application. It allows new users to sign up by providing their details, such as name, username, email, password, and telephone number. The code ensures that all inputs are valid, that there are no duplicate usernames or emails, and that passwords match before storing the new user information in the database.

2. Functionalities

1. **Input Validation**: Ensures that the data entered by the user is clean and meets the required format.
2. **Duplicate Checks**: Verifies that the username and email provided by the user are not already in use.
3. **Password Hashing**: Encrypts the password before storing it in the database for security reasons.
4. **Database Insertion**: Inserts the new user's information into the database if all validations pass.

5. **Feedback Messaging**: Provides feedback to the user about the registration status or any errors that occurred.

3. How Does the Code Work?

1. **Including Required Files**: The code starts by including files needed for session management and database connection.
2. **Defining Functions**:
    - **Input Validation Function**: A function is defined to clean and validate various types of input. Depending on the type of input (e.g., string, email, telephone), different validation rules are applied. This ensures that inputs are in the correct format before being used.
    - **User Existence Checks**: Two functions are defined to check if a username or email already exists in the database. These functions prepare and execute SQL queries to search for the provided username or email in the users table.
3. **Handling Form Submission**:
    - When the registration form is submitted (indicated by the presence of register_user in the POST request), the input values are validated using the previously defined function.
    - The passwords are checked to ensure they match, and the telephone number is validated to meet specific formatting rules (starting with 0 and not having 8 or more consecutive identical digits).
    - The code then checks if the username or email already exists using the earlier defined functions. If any validation fails or if duplicates are found, a feedback message is set.
4. **Password Encryption**:
    - If all validations pass, the password is hashed using a secure hashing algorithm before being stored in the database. This ensures that the actual password is not saved directly, which improves security.
5. **Inserting Data into the Database**:
    - An SQL statement is prepared to insert the new user's data into the database. This includes the name, username, email, hashed password, and telephone number.
    - The data is bound to the SQL statement and executed. If the insertion is successful, a success message is set. If there is an error during this process, an error message is displayed.
6. **Redirecting and Messaging**:
    - After handling the registration process, the feedback message (whether successful or an error) is saved in the session. The user is then redirected to the registration page where this feedback message can be displayed.
7. **Closing Resources**:

- Finally, the database connection is closed to free up resources.

createReport.php

1. Purpose of the Code

The purpose of the code is to manage and display an inventory report for a web application. It provides users with an overview of their inventory, including total items, item types, categories, and the total value of the inventory. Additionally, the code supports pagination to manage large lists of items and includes options to export the inventory data as a CSV file or view graphical representations of the data.

2. Functionalities

1. **User Authentication Check**: Ensures that only logged-in users can access the page.
2. **Data Retrieval**: Gathers various statistics about the inventory, such as the total number of items, item types, categories, and the total inventory value.
3. **Pagination**: Allows the display of a manageable number of items per page and provides navigation controls to browse through pages.
4. **CSV Export**: Offers functionality to export the inventory data to a CSV file.
5. **Display and Navigation**: Displays inventory data in a table format and provides pagination controls for navigating through multiple pages of items.
6. **Additional Actions**: Provides options for viewing graphical representations of the data.

3. How Does the Code Work?

1. **Session Management and Authentication**:
    - The code begins by including a session management script to handle user sessions.
    - It checks if the user is logged in by verifying the presence of a session variable. If the user is not logged in, they are redirected to the login page.
2. **Database Connection**:
    - A connection to the database is established using a connection script.
3. **Data Retrieval**:
    - Several SQL queries are executed to fetch different pieces of information about the inventory:
        - The total number of item types.
        - The total number of items in the inventory.

- The total number of categories.
- The total value of all inventory items (calculated as the sum of the price multiplied by the quantity for each item).

4. **Pagination Setup**:
    - Pagination settings are determined based on user input. If no input is provided, default values are used.
    - The current page and the number of items to display per page are retrieved from the form submission.
    - The appropriate SQL query is executed to fetch items for the current page, with a limit set by the pagination parameters.
    - The total number of pages is calculated based on the total number of items and the number of items per page.
5. **CSV Export Handling**:
    - If the user chooses to export data as a CSV file, headers are set for CSV content.
    - An SQL query is executed to retrieve all inventory items, and the results are written to a CSV file using PHP's built-in functions.
    - The file is then sent to the browser for download.
6. **HTML Display**:
    - The HTML section of the code displays an overview of the inventory statistics in a visually appealing manner using cards.
    - A table is used to list inventory items, and pagination controls are provided to navigate through the pages of items.
    - Forms are included for selecting the number of items per page, exporting to CSV, and viewing graphs.
7. **Additional Buttons**:
    - Buttons are provided for exporting data to a CSV file and viewing graphical representations of the inventory.
8. **Resource Cleanup**:
    - At the end of the script, the database connection is closed to release resources.

DBconnection.php

1. Purpose of the Code

The purpose of this code is to establish a connection to a MySQL database using PHP. This connection is crucial for the web application to interact with the database, enabling it to perform operations such as retrieving, updating, and storing data.

2. Functionalities

1. **Define Database Configuration**: Sets up the necessary details required to connect to the MySQL database.
2. **Create Database Connection**: Establishes a connection to the database using the provided configuration.
3. **Check Connection**: Verifies if the connection was successful and handles any connection errors.

3. How Does the Code Work?

1. **Database Configuration**:
    - The code begins by defining constants for the database connection. These constants include:
        - **DB_HOST**: The hostname of the MySQL server, usually "localhost" if the database is hosted on the same server as the web application.
        - **DB_USER**: The username used to access the MySQL database.
        - **DB_PASSWORD**: The password associated with the MySQL username.
        - **DB_NAME**: The name of the database to connect to.
2. **Create Database Connection**:
    - The mysqli class is used to create a connection to the MySQL database. An instance of mysqli is created using the defined constants for the host, user, password, and database name. This instance is assigned to the variable $conn.
3. **Check Connection**:
    - The code checks if the connection to the database was successful by evaluating if there was an error during the connection process.
    - If there was an error (for example, if the hostname, username, or password was incorrect), an error message is displayed using the die function. This function terminates the script and shows the connection error message.

delete_account.php

1. Purpose of the Code

The purpose of this code is to handle the deletion of a user account from the web application. It ensures that a logged-in user can delete their account and be redirected appropriately based on the success or failure of the deletion operation.

2. Functionalities

1. **Session Management**: Ensures that only logged-in users can perform account deletion.
2. **Account Deletion**: Deletes the user's account from the database if requested.
3. **Feedback Handling**: Provides feedback to the user if the account deletion fails.
4. **Session and Redirection**: Ends the user's session and redirects them to the login page after successful deletion.

3. How Does the Code Work?

1. **Session Management**:
    - The code starts by including a script (session.php) that manages user sessions.
    - It then includes another script (DBconnection.php) to establish a connection to the database.
    - The script checks if the user is logged in by verifying the presence of a session variable named loggedIn. If the user is not logged in, they are redirected to the login page.
2. **Retrieve User Information**:
    - The user's full name is retrieved from the session and stored in the variable $fullName.
3. **Handle Account Deletion**:
    - The script checks if the request method is POST and if the delete_account button has been pressed. This indicates that the user wants to delete their account.
    - A SQL query is prepared to delete the user from the database where the name matches the $fullName variable.
    - The query is executed using a prepared statement to ensure security and prevent SQL injection.
4. **Post-Deletion Actions**:
    - If the deletion is successful, the user's session is destroyed using session_destroy(), and they are redirected to the login page.
    - If the deletion fails, a feedback message is set in the session to inform the user of the failure, and they are redirected back to the account page (my_account.php).

5. **Close Database Connection**:
    - Finally, the database connection is closed to free up resources

deletedata.php

1. Purpose of the Code

The purpose of this code is to handle the deletion of items and categories from an inventory management system. It ensures that items and categories can be deleted from the database, and provides feedback on the success or failure of these operations.

2. Functionalities

1. **Delete an Item**: Allows the deletion of an item based on its ID.
2. **Delete a Category**: Allows the deletion of a category, but only if there are no items linked to that category.
3. **Feedback Message**: Provides feedback to the user about the result of the deletion operations.
4. **Redirection**: Redirects the user to a management page with a feedback message.

3. How Does the Code Work?

1. **Include Database Connection**:
    - The code starts by including the DBconnection.php file to establish a connection to the MySQL database.
2. **Initialize Feedback Message**:
    - A variable named $feedback_message is initialized to store feedback messages for the user.
3. **Handle Item Deletion**:
    - **Check for Item ID**: The code checks if the delete_item_id parameter is set in the POST request. This parameter represents the ID of the item to be deleted.
    - **Validate Item ID**: The item ID is validated to ensure it is a positive integer. If valid, a prepared SQL statement is used to delete the item from the database.
    - **Execute and Check Deletion**: If the deletion is successful and at least one row was affected, a success message is set. If no rows were affected, it indicates that no item with the given ID was found.
    - **Handle Errors**: If the SQL statement preparation or execution fails, an appropriate error message is set.
4. **Handle Category Deletion**:

- **Check for Category ID**: The code checks if the delete_category_id parameter is set in the POST request. This parameter represents the ID of the category to be deleted.
- **Validate Category ID**: The category ID is validated to ensure it is a positive integer. If valid, it first checks if there are any items linked to this category.
- **Check Linked Items**: A query is prepared and executed to count items associated with the category. If items are found, a message is set indicating that the category cannot be deleted because it is still in use.
- **Delete Category**: If no items are linked to the category, another SQL statement is prepared to delete the category from the database.
- **Execute and Check Deletion**: If the deletion is successful and at least one row was affected, a success message is set. If no rows were affected, it indicates that no category with the given ID was found. Error handling is included for SQL statement preparation and execution failures.

5. **Close Database Connection**:
   - The database connection is closed to release resources.
6. **Redirect with Feedback**:
   - The user is redirected to the manage_inventory.php page with the feedback message encoded in the URL. This provides the user with information about the result of the deletion operation.

get_item.php

1. Purpose of the Code

The purpose of this code is to retrieve and return detailed information about a specific item from the inventory, along with a list of all available categories. This information is provided in a structured format (JSON) suitable for use by a client-side application, such as a web interface that needs to display or update item details.

2. Functionalities

1. **Retrieve Item Details**: Fetches detailed information about an item based on its ID.
2. **Fetch Categories**: Retrieves a list of all categories available in the inventory.
3. **Return Data in JSON Format**: Provides the item details and categories in JSON format for use by client-side applications.
4. **Error Handling**: Handles and returns errors related to invalid IDs or database issues.

3. How Does the Code Work?

1. **Include Database Connection**:
    - The code starts by including the DBconnection.php file to connect to the database.
2. **Sanitize and Validate ID**:
    - The ID of the item to be retrieved is obtained from the GET parameters. It is sanitized and validated to ensure it is a positive integer. If the ID is invalid, an error message is returned.
3. **Fetch Item Details**:
    - **Prepare SQL Query**: A SQL query is prepared to select the item's details from the database based on the provided ID.
    - **Execute Query**: The query is executed with the item ID as a parameter.
    - **Handle Query Errors**: If the SQL statement preparation fails, an error message is returned, and the script exits.
    - **Retrieve Results**: The result of the query is fetched, and item details are stored in an array.
    - **Close Statement**: The prepared statement is closed.
4. **Fetch Categories**:
    - **Prepare SQL Query for Categories**: Another SQL query is prepared to fetch all categories from the database.
    - **Execute Query**: The query is executed to retrieve the list of categories.
    - **Handle Query Errors**: If this SQL statement preparation fails, an error message is returned, and the script exits.
    - **Retrieve Results**: The result is fetched and all categories are stored in an array.
    - **Close Statement**: The prepared statement for categories is closed.
5. **Return Data**:
    - **Check for Item Existence**: If the item is found, its details along with the list of categories are encoded into JSON format and returned.
    - **Handle Item Not Found**: If no item is found with the given ID, an error message indicating "Item not found" is returned.
    - **Handle Invalid ID**: If the ID is not valid, an error message is returned.
6. **Close Database Connection**:
    - The database connection is closed to free up resources.

graph.php

1. Purpose of the Code

The purpose of this code is to provide an overview of the inventory system in the web application. It displays key statistics and visualizes data about the inventory, such as the total number of items, item types, categories, and the total value of items. It also generates a chart that shows the total quantity of items for each category. This information helps users understand the state of their inventory and provides insights into how inventory is distributed across different categories.

2. Functionalities

1. **Session Management**:
    - Ensures that only logged-in users can access the page. If the user is not logged in, they are redirected to the login page.
2. **Database Queries**:
    - Retrieves the total number of items, the total quantity of items, the total number of categories, and the total value of all items from the database.
    - Fetches the total quantity of items grouped by their categories for visualization.
3. **Data Visualization**:
    - Creates a bar chart to show the total quantity of items in each category.
4. **Display Information**:
    - Presents key statistics and a chart on a web page using Bootstrap for styling and CanvasJS for the chart.
5. **Navigation**:
    - Provides a button to navigate back to a previous page for creating reports.

3. How Does the Code Work?

1. **Session Check**:
    - The code begins by including a session management script to ensure that the user is logged in. If the user is not authenticated, they are redirected to the login page.
2. **Database Connection**:
    - The database connection is established by including the DBconnection.php file, which contains the database credentials and connection setup.
3. **Retrieve Statistics**:
    - **Total Item Types**: A query is executed to count the total number of items in the inventory.

- **Total Inventory Quantity**: A query calculates the sum of quantities for all items.
- **Total Categories**: A query counts the number of categories available.
- **Total Value**: A query calculates the total value of all items by multiplying their price by quantity and summing these values.

4. **Generate Data for Chart**:
   - A query is executed to calculate the total quantity of items for each category.
   - The results are stored in an array called dataPoints, which is used to create a bar chart.
5. **Display Web Page**:
   - **Chart Rendering**: The chart is created using the CanvasJS library. JavaScript is used to render the chart with data passed from PHP.
   - **Information Display**: Bootstrap is used to style and display key statistics (e.g., total items, categories, and value) in cards on the page.
   - **Alert Message**: An alert is shown to notify users that categories with no items are not included in the chart.
6. **Back Button**:
   - A button is provided to navigate back to the report creation page. JavaScript handles the button click event to redirect the user to createReport.php.
7. **Close Database Connection**:
   - The database connection is closed after all queries have been executed to free up resources.

home.php

1. Purpose of the Code

The purpose of this code is to create a welcome page for an inventory management system. It greets the user, showcases the features of the application, and provides a visual representation of the system's capabilities. This page aims to provide a user-friendly introduction to the system, including a loader animation for visual appeal and a section outlining the key services offered by the application.

2. Functionalities

1. **User Authentication**:
   - The page ensures that only logged-in users can access it. If the user is not logged in, they are redirected to the login page.
2. **Welcome Message**:

- Displays a personalized welcome message that includes the user's name.

3. **Loader Animation**:
   - Shows an animated loader that simulates a truck moving on a road. This animation adds a dynamic visual element to the page.
4. **Service Information**:
   - Provides information about the main features of the inventory system through a series of service cards. Each card highlights a specific feature, such as managing inventory, searching for items, and generating reports.
5. **Responsive Design**:
   - Ensures that the page is visually appealing and functional on different screen sizes by using responsive design techniques.
6. **Footer**:
   - Includes a footer section that is required from another file, adding consistent navigation or information at the bottom of the page.

3. How Does the Code Work?

1. **Session Management**:
   - The code starts by including a session management script to handle user authentication. If the user is not logged in, they are redirected to the login page. The user's name is retrieved from the session and safely displayed on the page.
2. **Style Definition**:
   - CSS styles are defined to manage the appearance of the loader animation, truck graphics, and other page elements. This includes the animation of a truck moving up and down, a moving road, and a lamp post. These styles are embedded directly in the HTML within a <style> tag.
3. **Page Content**:
   - The HTML body is structured into sections:
     - **Hero Section**: Contains a welcome message for the user, personalized with the user’s name. It also includes a loader animation that features a truck moving along a road, designed using SVG graphics and CSS animations.
     - **Services Section**: Showcases the main features of the application using cards. Each card contains an image and a description of a feature, such as inventory management, search functionality, and report creation.
4. **Footer Inclusion**:

- A footer is included by requiring an external PHP file (Components/footer.php). This ensures that the footer is consistently displayed across different pages.

5. **Responsive Design**:
    - Media queries are used to adjust font sizes and other design elements based on the screen width, ensuring that the page remains usable and attractive on devices of various sizes.

login.php

1. Purpose of the Code

The purpose of this code is to create a login page for an inventory management system. It is designed to allow users to enter their credentials to access the system. If a user is already logged in, they are redirected to the home page. The login page provides a form where users can input their username and password, and it also handles displaying error messages if the login fails.

2. Functionalities

1. **User Redirection**:
    - If a user is already logged in, they are automatically redirected to the home page, preventing them from accessing the login page again.
2. **Error Messaging**:
    - Displays an error message if there is a problem with the login attempt, such as incorrect username or password.
3. **Login Form**:
    - Provides a form for users to enter their username and password. This form sends the entered information to a script that checks the credentials.
4. **Back Button**:
    - Includes a "Back" button that allows users to navigate back to the home page.
5. **Responsive Design**:
    - Ensures that the login page is displayed correctly on various screen sizes, including mobile devices.

3. How Does the Code Work?

1. **Session Management**:

- The code begins by including a session management script. This script checks if a user is already logged in by looking at the session data. If the user is logged in, they are redirected to the home page to prevent accessing the login page again.

2. **Error Message Handling**:
   - If an error occurs during login (e.g., incorrect credentials), an error message is retrieved from the URL query string. This message is then displayed on the login page using an alert box. The message is safely displayed by converting any special characters into a harmless format.
3. **Page Layout and Styling**:
   - The page layout includes a navigation bar at the top, a main section containing the login form, and a footer. CSS styles are used to ensure the layout looks good on different devices and screen sizes.
4. **Login Form**:
   - The form is designed to collect the user's username and password. It uses HTML input fields with validation rules to ensure that the data entered is in the correct format. For example, the username field allows only letters and numbers, while the password field must meet specific requirements for security.
5. **Back Button Functionality**:
   - A "Back" button is included in the navigation bar. When clicked, it redirects the user to the home page. This is achieved using JavaScript, which listens for a click event on the button and then changes the browser's location to the home page URL.
6. **Responsive Design**:
   - CSS styles are applied to ensure that the login page adjusts to different screen sizes. For instance, adjustments are made for small screens to ensure the content remains accessible and well-organized.

logout.php

1. Purpose of the Code

The purpose of this code is to log a user out of the inventory management system. It ensures that all user-specific session data is cleared and that the user is redirected to the login page. This is typically used when a user wants to end their session and securely exit the application.

2. Functionalities

1. **Clearing Session Data**:
   - All session variables are cleared to remove any stored user information.

2. **Destroying the Session**:
    - The session itself is terminated to ensure that it is completely ended.
3. **Redirecting to Login Page**:
    - After logging out, the user is redirected to the login page.

3. How Does the Code Work?

1. **Including Session Management Script**:
    - The code starts by including a script (session.php) that handles session management. This script is necessary to properly manage and interact with user sessions.
2. **Clearing Session Data**:
    - session_unset() is called to remove all session variables. This effectively clears any user-specific data that was stored during the session, such as login status or user preferences.
3. **Destroying the Session**:
    - session_destroy() is called next. This function terminates the session, ensuring that the session file on the server is deleted. This step is crucial for ensuring that no session data remains accessible after the user has logged out.
4. **Redirecting to Login Page**:
    - Finally, the header('Location: index.php') function is used to send a redirection header to the browser. This instructs the browser to navigate to the login page (index.php). The exit() function is then used to stop further script execution, ensuring that no additional code is run after the redirection.

manage_inventory.php

1. Purpose of the Code

The purpose of this code is to create a page for managing an inventory within a web application. This page allows users to:

- **Add New Items**: Users can input details about new inventory items and submit them to be added to the system.
- **Manage Categories**: Users can add new categories or delete existing ones from the inventory.
- **View and Manage Existing Items**: Users can see a list of all inventory items, including their details, and perform actions such as editing or deleting them.

- **Provide Feedback**: Display messages to inform users about the success or failure of their actions.

2. Functionalities

1. **User Authentication Check**:
    - Ensures that only logged-in users can access the page. If the user is not logged in, they are redirected to the login page.
2. **Database Interaction**:
    - Retrieves and displays categories for item addition and manages the items list with pagination.
3. **Feedback Messaging**:
    - Displays messages based on the success or failure of actions like adding or deleting items.
4. **Item Management**:
    - **Add New Item**: A form is provided to add new items to the inventory with fields for name, description, price, quantity, and category.
    - **Manage Categories**: Forms for adding and deleting categories are included.
    - **Display Items**: Shows a table of existing items with options to edit or delete each item.
5. **Pagination**:
    - Allows users to navigate through multiple pages of inventory items.
6. **Update Functionality**:
    - Users can edit existing items through a modal form that is dynamically populated with the item's current details.
7. **Responsive Design**:
    - Ensures the page layout is adaptable to various screen sizes and devices.

3. How Does the Code Work?

1. **User Authentication**:
    - **Session Management**: The code includes a session management script that checks if a user is logged in. If the $_SESSION['loggedIn'] variable is not set, indicating that the user is not authenticated, they are redirected to the login page using the header('Location: login.php') function.
2. **Database Connection**:

- **Database Query**: A database connection script (DBconnection.php) is included, and queries are executed to fetch categories and items. The results are used to populate forms and display data.

3. **Feedback Message Handling**:
    - **Retrieve Feedback**: The feedback message from URL parameters is retrieved and displayed as an alert. The type of alert (info, warning, success, or danger) is determined based on the content of the feedback message.
4. **Add New Item**:
    - **Form Submission**: A form is provided for adding new items, including fields for name, description, price, quantity, and category. When the form is submitted, the data is sent to adddata.php for processing.
5. **Manage Categories**:
    - **Add Category**: A form allows users to add a new category, which is submitted to adddata.php.
    - **Delete Category**: Another form allows users to select and delete a category. This data is sent to deletedata.php.
6. **Display and Manage Items**:
    - **Pagination Setup**: The number of items per page and the current page are set based on URL parameters. Items are fetched from the database with limits and offsets according to the current page and items per page.
    - **Items Table**: A table displays all items, including their ID, name, description, quantity, price, and category. Each item has buttons for editing and deleting.
7. **Update Modal**:
    - **Edit Item**: An update modal form is used for editing item details. JavaScript is used to fetch item data via an AJAX request (get_item.php) and populate the modal fields. When the form is submitted, the data is sent to updatedata.php for processing.
8. **AJAX for Update**:
    - **Fetch Item Data**: JavaScript fetches item details from the server and populates the update modal.
    - **Form Submission**: The update form uses AJAX to send the updated data to the server without reloading the page. Feedback is displayed based on the response.
9. **Responsive Design**:
    - **CSS and Bootstrap**: The page uses CSS and Bootstrap classes to ensure a responsive layout that adjusts to different screen sizes.

my_account.php

1. Purpose of the Code

The purpose of the code is to create a user profile management page for a web application. This page allows users to view and update their profile information, such as their name, email, and phone number. Users can also change their password and delete their account if needed. The code ensures that only logged-in users can access this page and handle their profile data securely.

2. Functionalities

1. **User Authentication Check**: The code checks if a user is logged in. If a user is not logged in, they are redirected to the login page.
2. **Fetch User Data**: It retrieves the user's profile information from the database based on the user's session data.
3. **Profile Update**: It allows users to update their profile details like name, email, phone number, and password. The password can be left empty if the user does not want to change it.
4. **Account Deletion**: It provides an option for users to delete their account. When this option is selected, the user's data is removed from the database, and the user is logged out and redirected.
5. **Form Submission**: The code handles form submissions for updating profile details and for account deletion.
6. **Feedback Messages**: It shows feedback messages to inform users about the success or failure of their actions, such as profile updates or account deletions.

3. How Does the Code Work?

1. **Session Management**: The code begins by including a script that handles session management. This script checks if the user is logged in by looking at session variables. If the user is not logged in, they are redirected to the login page.
2. **Database Connection**: A connection to the database is established using a separate file. This connection is necessary for fetching and updating user data.
3. **Fetching User Data**: The user’s full name is used to fetch their profile information from the database. This data includes the username, email, and phone number. The fetched data is then displayed on the profile page.
4. **Handling Form Submission**: When the form is submitted, the code first checks if the "Delete Account" button was clicked. If it was, the user's account is deleted from the database, and their session is destroyed, logging them out and redirecting them to the homepage.
5. **Profile Update**: If the form is used to update profile details, the code processes the input data. Each field (name, email, phone number, and password) is validated for correctness. If the data

is valid, the code updates the user's information in the database. For password changes, the new password is hashed (encrypted) before being stored to ensure security.

6. **Feedback Messages**: After processing the form, the code sets a feedback message based on whether the update or deletion was successful or if there were any errors. These messages are then displayed to the user.
7. **HTML and CSS Styling**: The HTML section of the code is responsible for displaying the profile management form. CSS styles are applied to make the page look appealing and to ensure that the form is easy to use. This includes styling for buttons, form fields, and layout adjustments for different screen sizes.
8. **JavaScript for Account Deletion**: JavaScript is used to handle the account deletion process. When the "Delete Account" button is clicked, a confirmation dialog appears to ensure the user really wants to delete their account. If confirmed, a hidden input is added to the form to indicate that account deletion is requested, and then the form is submitted.

register.php

1. Purpose of the Code

The purpose of this code is to create a sign-up page for a web application. This page allows new users to register by providing their personal information. The sign-up form collects data such as name, username, email, telephone number, and password. Feedback is provided to the user based on the outcome of their registration attempt.

2. Functionalities

1. **User Feedback**: Displays messages to the user about the result of their registration attempt, such as success or errors.
2. **Form Fields**: Collects user data through various input fields like name, username, email, phone number, and password.
3. **Form Validation**: Ensures that the information entered by the user meets specific requirements before submission.
4. **Form Submission**: Sends the user's data to a server-side script for processing when the form is submitted.
5. **Navigation Button**: Provides a button to navigate back to the homepage.
6. **Visual Elements**: Includes images and styling to enhance the appearance of the sign-up page.

3. How Does the Code Work?

1. **Including Components**: At the start, the header component of the page is included. This ensures that common elements like the page header are consistently displayed. Another file,

checkSignup.php, is included to handle the registration logic, such as validating and saving user data.

2. **Feedback Handling**: The code checks if there is any feedback message stored in the session. This message is retrieved and then cleared from the session to avoid displaying the same message again on page reloads. The feedback message helps inform the user about the status of their registration (e.g., whether it was successful or if there were errors).
3. **Navigation Bar**: A navigation bar is created at the top of the page. It includes the application's logo and a button that, when clicked, redirects users back to the homepage.
4. **Main Content**: The main content of the page is enclosed in a section that has a background color. Inside this section:
5. **Feedback Alert**: If there is a feedback message, an alert box is displayed with the appropriate class (e.g., success, warning) based on the content of the message. This alert box provides feedback to the user about their registration attempt.
6. **Sgn-Up Form**: A form is presented where users can enter their details. The form fields include:
    - **Name**: Users enter their name.
    - **Username**: Users choose a unique username.
    - **Email**: Users provide a valid email address.
    - **Telephone Number**: Users enter a 10-digit phone number.
    - **Password**: Users create a password that meets specific criteria (e.g., including numbers and letters).
    - **Confirm Password**: Users repeat their password to ensure it was entered correctly.
    - **Terms and Conditions**: Users must agree to the service agreement by checking a checkbox.
7. **Form Submission**: When the form is submitted, it sends the data to the checkSignup.php file for processing. The register_user hidden input field helps identify that this form is for user registration.
8. **Visual Enhancement**: An image is included for visual appeal, making the sign-up page more engaging.
9. **Back Button**: A button labeled "Back" is provided for users to navigate to the homepage. JavaScript is used to handle the button click event, which redirects users to the homepage when clicked.
10. **Including Footer**: Finally, the footer component is included at the bottom of the page, ensuring that common footer elements are displayed.

search.php

1. Purpose of the Code

The purpose of this code is to create a web page that allows users to search for items in an inventory system. Users can search by item ID, name, or category, view the results, and navigate through multiple pages of results. The page also includes options to set the number of items displayed per page and to clear search criteria.

2. Functionalities

1. **User Authentication**: Users are redirected to the home page if they are not logged in, ensuring that only authorized users can access the search functionality.
2. **Search Functionality**: Users can search for items by specifying criteria such as item ID, name, or category.
3. **Pagination**: The page displays search results in multiple pages, allowing users to navigate through them.
4. **Dynamic Content Display**: Results and feedback messages are shown based on the user's search input and the query results.
5. **Clear Search Criteria**: Users can clear their search criteria to start a new search.
6. **Feedback Messages**: Inform users about the status of their search, including errors or no results found.

3. How Does the Code Work?

1. **Session Management**: The code starts by including a script that manages user sessions. If a user is not logged in, they are redirected to the home page to prevent unauthorized access.
2. **Database Connection**: A connection to the database is established to retrieve item data based on the user's search criteria.
3. **Search Parameters and Pagination**:
    - Variables are initialized to store search parameters (such as item ID, name, and category) and pagination settings (like items per page and current page).
    - The number of items displayed per page and the current page number are used to calculate an offset for fetching the correct set of items.
4. **Search Query Preparation**:
    - Depending on the search parameters provided by the user, different SQL queries are prepared.
    - If an item ID is provided, the query will search by ID.

- If a name or category is provided, the query will search using the LIKE operator to match partial names or categories.

5. **Executing the Search**:
   - The appropriate SQL query is executed based on the search criteria.
   - Parameters are bound to the query to protect against SQL injection.
   - The results of the query are fetched to display items that match the search criteria.
6. **Counting Total Items**:
   - Another query is executed to count the total number of items that match the search criteria. This count is used to determine how many pages of results are needed for pagination.
7. **Displaying Results**:
   - If results are found, they are displayed in a table format.
   - Pagination controls are included to navigate through different pages of search results.
8. **Search and Clear Form**:
   - A form allows users to enter search criteria and submit their search.
   - A separate button is provided to clear all search fields and submit the form to reset the search.
9. **Pagination Controls**:
   - Controls for pagination are included, allowing users to navigate to previous or next pages of results and select specific pages.
10. **Feedback Messages**:
    - Feedback messages are displayed based on the search results. If no items are found or if there is a general message, it is shown to the user in an alert box.
11. **Clear Button Functionality**:
    - JavaScript is used to clear all search fields when the "Clear" button is clicked. After clearing the fields, the form is automatically submitted to update the search criteria.
12. **Footer and Database Connection Closure**:
    - The footer of the page is included for consistent styling.
    - Finally, the database connection is closed to free up resources.

session.php

1. Purpose of the Code

The purpose of this code is to configure and start a PHP session securely. Sessions are used to store user-specific data across different pages of a web application, such as user login information or shopping cart contents. This code ensures that the session is handled securely by setting various session cookie parameters and enforcing strict session management practices.

2. Functionalities

1. **Session Cookie Configuration**: Sets parameters for how session cookies are handled.
2. **Strict Session Management**: Enforces strict mode to enhance security.
3. **Session Initialization**: Starts a new session or resumes an existing one.
4. **Session ID Regeneration**: Regenerates the session ID to protect against session fixation attacks.

3. How Does the Code Work?

1. **Session Cookie Configuration**:
    - The session_set_cookie_params function is used to define settings for the session cookie, which is stored on the user's browser. This cookie is essential for maintaining the session between different pages.
    - lifetime is set to 0, meaning the cookie will expire when the browser is closed.
    - path is set to /, making the cookie available across the entire website.
    - domain is left empty, which means the cookie is valid for the current domain.
    - secure is set to true, ensuring the cookie is only sent over HTTPS connections. This helps prevent the cookie from being intercepted by attackers.
    - httponly is set to true, which prevents JavaScript from accessing the cookie. This helps protect against certain types of attacks.
    - samesite is set to Strict, which helps prevent Cross-Site Request Forgery (CSRF) attacks by only allowing cookies to be sent with requests originating from the same site.
2. **Strict Session Management**:
    - The ini_set('session.use_strict_mode', 1) function enforces strict mode for session management. This mode helps prevent the use of uninitialized sessions, which could be a security risk.
3. **Session Initialization**:

- session_start() is called to begin a session or resume an existing one. This must be called before any output is sent to the browser and is necessary for working with session variables.

4. **Session ID Regeneration**:
   - session_regenerate_id(true) is used to generate a new session ID. This is done to prevent session fixation attacks, where an attacker could try to hijack an existing session by setting a session ID that they know. Regenerating the session ID ensures that the user's session is secure and unique.

SLA.php

1. Purpose of the Code

The purpose of this code is to create a webpage that displays the Service Level Agreement (SLA) for a web application. An SLA is a document that outlines the expected performance and service standards for the application. This page details various metrics that define how well the application should perform, including uptime, page load times, data accuracy, and more. It also explains the penalties for not meeting these standards.

2. Functionalities

1. **Header**: Displays the title of the SLA document at the top of the page.
2. **SLA Sections**: Each section describes a different aspect of the SLA, including system availability, page load times, data accuracy, report generation time, customer support response time, and system update downtime.
3. **Back Button**: Provides a link to return to the main page of the application.
4. **Footer**: Contains additional notes about the SLA, including information about penalties, reporting, and review schedules.

3. How Does the Code Work?

1. **Header and Basic Styling**:
   - The <header> tag contains the main title of the SLA page. It is styled to have a dark background with white text and centered alignment.
   - Basic styling is applied to reset default browser styles and set up the font, colors, and layout. The body has a light background color and a default text color for better readability.
2. **SLA Sections**:

- Each SLA section is enclosed in a <section> tag with a class of sla-section. This class applies styling to each section, giving it a white background, padding, a border radius, and a subtle shadow to make it stand out.
- Inside each section, a with the class container ensures that the content is properly aligned and does not exceed a maximum width.
- Each SLA section contains a heading (<h2>) and several paragraphs (<p>) or lists (<ul>) describing the specific SLA metrics. These sections provide details on targets, measurement methods, allowed deviations, and penalties.

3. **Responsive Design**:
   - A media query is used to adjust the layout for smaller screens (e.g., mobile devices). For screens with a maximum width of 768 pixels, padding in the header and footer is reduced, and the font size of headings is adjusted to ensure better readability on smaller devices.
4. **Back Button**:
   - A button with the class btn and btn-outline-primary is included in a with the class btn-container. This button links back to the main page (index.php). It is centered and provides an easy way for users to navigate back.
5. **Footer**:
   - The <footer> tag contains important notes about the SLA. It explains that penalties are cumulative but capped, provides information about performance reports, and mentions that the SLA will be reviewed quarterly.

updatedata.php

1. Purpose of the Code

The purpose of this code is to update the details of an item in the inventory database. This involves changing the item's information such as its name, description, price, quantity, and category based on user input. The code ensures that the provided data is sanitized and validated before making changes to the database.

2. Functionalities

1. **Input Handling**: The code processes data submitted through a POST request, which typically occurs when a user submits a form.
2. **Data Sanitization**: Input data is cleaned to prevent security issues such as SQL injection or cross-site scripting (XSS).
3. **Data Validation**: The code checks if all required fields are provided and valid.

4. **Database Update**: An SQL query is prepared and executed to update the item details in the database.
5. **Feedback Messages**: Users are informed about the success or failure of the update operation.

3. How Does the Code Work?

1. **Database Connection**:
    - The code starts by including a file (DBconnection.php) which is expected to establish a connection to the database. This connection is used to execute SQL queries.
2. **Handling the POST Request**:
    - The code checks if the request method is POST, which means that the form data was submitted. If not, it outputs an error message indicating an invalid request method.
3. **Sanitizing Input Data**:
    - Data from the form is retrieved and sanitized using functions that ensure the input is safe to use. For instance:
        - **FILTER_SANITIZE_NUMBER_INT**: Removes any non-numeric characters from the ID and quantity fields.
        - **FILTER_SANITIZE_FULL_SPECIAL_CHARS**: Converts special characters to HTML entities to prevent XSS attacks.
        - **FILTER_SANITIZE_NUMBER_FLOAT**: Ensures the price field contains only valid floating-point numbers.
4. **Validating Data**:
    - The code verifies that all required fields are provided and are not empty or invalid. If any of the fields are invalid, an error message is shown, and the script exits.
5. **Preparing and Executing the SQL Query**:
    - A SQL UPDATE statement is prepared to modify the existing record in the database based on the item ID. The prepare function is used to prevent SQL injection attacks by using placeholders in the query.
    - The bind_param function binds the sanitized input data to the SQL query. This ensures that the data is correctly inserted into the query.
    - The query is executed with execute, and the number of affected rows is checked to determine if the update was successful.
6. **Providing Feedback**:
    - After executing the query, the code checks if any rows were affected. If so, a success message is shown. If no rows were affected, it means that either the item was not found or no actual changes were made, and an appropriate message is displayed.

- If the update fails for any reason, an error message is shown.

7. **Closing Connections**:
    - The prepared statement and database connection are closed to free up resources.